Preprint ANL/MCS-P4068-0413

Real-time Stochastic Optimization of Complex Energy Systems on High Performance Computers

Cosmin G. Petra
Argonne National Laboratory,
9700 S. Cass Avenue
Argonne, IL 60439
USA
petra@mcs.anl.gov

Olaf Schenk
Institute of Computational Science,
Faculty of Informatics,
Università della Svizzera italiana,
Switzerland
olaf.schenk@usi.ch

Mihai Anitescu
Argonne National Laboratory,
9700 S. Cass Avenue
Argonne, IL 60439
USA
anitescu@mcs.anl.gov

***Abstract*—We present a scalable approach that computes in operationally-compatible time the energy dispatch under uncertainty for complex energy systems of realistic size. Complex energy systems, such as the US power grid, are affected by increased uncertainty of its target power sources, due for example to increasing penetration of wind power coupled with the physical impossibility of very precise wind forecast. The leading optimization under uncertainty paradigm for such problems, stochastic programming, requires thousands of simultaneous scenarios, giving problems with billions of variables that need to be solved within an operationally defined time interval. To address this challenge, we propose several algorithmic and implementation advances inside our hybrid parallel optimization solver PIPS-IPM. The new developments include a novel incomplete augmented multicore sparse factorization implemented within PARDISO linear solver and new multicore- and GPU-based dense matrix implementations. We also adapt and improve the interprocess communication strategy. Numerical experiments on "Titan" (Cray XK7) and "Piz Daint" (Cray XC30) show that 24-hour horizon problems with thousands of scenarios can be efficiently solved in parallel in times compatible with the operational practices. To our knowledge, "real-time" compatible performance on a broad range of architectures for this class of problems has not been possible prior to present work.**

I. Introduction

In this paper, we present a scalable framework for solving two-stage stochastic optimization problems with recourse arising in the optimization of power grid under uncertainty. The problem we solve is the one of deciding the optimal operation of electricity generation facilities to produce energy at the lowest cost and to reliably serve consumers, recognizing any operational limits of generation and transmission facilities.

In the US, power grid optimization problems are solved by each of the 10 independent system operators (ISOs) [1]. In the form of unit commitment (UC), such problems are the main component of day-ahead planning of generators and electricity markets and they currently are solved faster than 1 hour [2]. In the form of economic dispatch (ED), these optimization problems are used to balance supply and demand, and they need to be solved within several minutes [2]. We note that these time windows reflect current practice only, and evolution of energy operations to include more renewable energy dispatch are likely to both increase the size of the problems and reduce the time horizon within which they need to be solved. The economic footprint of such problems is enormous; in the US, solving such problems results in dispatch orders to generators that are worth several billions to tens of billions of dollars per year per ISO, for a national total of hundreds of billions of dollars per year. Their critical contribution to the US economy has lead to their analysis as technologies being specifically controlled by law, for example in the Energy Policy Act of 2005, Sections 1298 and 1832.

In this paper, we focus on the computing challenges stemming from one such evolutionary imperative: accounting for the variability in energy supply availability that occurs when renewable energy source such as wind are used by using optimization under uncertainty techniques such as stochastic optimization [3], [4], [5]. This results in vastly larger optimizations problems, with several billions variables and constraints, because a large number of possible realizations of the uncertainty need to be considered to accurately capture the stochastic component of the problem. As the problem is very large and needs to be solved within restrictive time limits, a high-end distributed memory supercomputing solution is not only useful, it is also required. We have developed PIPS-IPM optimization solver which implements an interior-point methods and specialized sparse and dense linear algebra. PIPS-IPM's main computational bottleneck is the solution of linear systems that needs to be solved at each step. Several features of the problem beyond its large size create difficulties in achieving high performance when solving it. Namely,

(i) the linear system has hybrid sparse and dense features, stemming from the different nature of the two stages of the problem;
(ii) direct sparse matrix factorizations needs to be used because the ill-conditioning of the linear systems due to the use of interior-point methods makes iterative solvers ineffective both in terms of computational cost and accuracy of the solutions;
(iii) the constraint matrix is a mix of power-flow constraints, multiple technological constraints on the generators and time replication, which makes virtually impossible to tackle with sparsity in a structured way during sparse factorizations;

SC13, November 10–16, 2013, Denver, Colorado, USA
978-1-4673-0806-9/13/$31.00 ©2013 IEEE

(iv) the matrices of interest are saddle-point matrices, being symmetric and indefinite, and require advanced pivoting and non-trivial solution-refinement techniques to maintain numerical stability.

Our previous work used a BG/P platform and obtained more than 90% parallel efficiency on up to 80% of "Intrepid" BG/P of Argonne National Laboratory [6]. The study also identified the intranode sparse scenario computations as being the bulk of the execution time and the main barrier in solving power grid optimization problems in "real-time".

As a consequence we propose a novel algorithmic solution for the sparse scenario computations that is based on a incomplete augmented factorization approach [7] and achieves good peak performance despite the complicating features of the problem. As a result, the optimization problem can be solved under the "real-time" requirement of the application. We also depart significantly from our previous study in the treatment of the two main computational bottlenecks of our decomposition approach, namely dense linear algebra computations and communication. For this we present implementation improvements such as mixed CPU-GPU computations and a new communication-computation pattern that make efficient use of modern architectures such as Cray XK7 and Cray XC30 and facilitate very good parallel efficiencies.

Our large scale numerical experiments performed on "Titan" XK7 machine from Oak Ridge National Laboratory and "Piz Daint" XC30 machine from Swiss National Supercomputing Centre show that these developments of this paper make possible solving realistically-sized (24-hour horizon) with thousands of scenarios in times that are considerably under one hour. To the best of our knowledge, this has not been possible before. We also observe very good strong scaling efficiencies on both systems, 79.2% on "Piz Daint" and 87.0% on "Titan", despite the significant acceleration of the intranode computations. The largest power grid optimization problem we solved in this work is a 24-hour horizon unit commitment problem with $16,384$ scenarios that has 1.95 billion variables and 1.947 billion constraints. On $16,384$ nodes of "Titan" (87.6% of the machine) we solved sparse indefinite linear systems of size as large as 7.8 billion.

The outline of the paper is as follows. In Section II we discuss the context of our work within the optimization of power grid systems. In Section III we describe our decomposition framework and in Section IV we present the new algorithmic advances and implementation developments. The numerical experiments as well as discussions of the parallel performance under different efficiency metrics are presented in Section V. We present our conclusions in Section VI.

II. Motivating Application: Stochastic Unit Commitment for Power Grid Systems

In our analysis, we consider a two-stage stochastic optimization formulation for stochastic unit commitment [2]. The problem has the following structure (c.f. [6]):

$$\min \left(\sum_{k=0}^{T} \sum_{j \in \mathcal{G}} f_j \cdot x_{k,j} \right) + \frac{1}{N} \sum_{s \in \mathcal{N}} \left(\sum_{k=0}^{T} \sum_{j \in \mathcal{G}} c_j \cdot G_{s,k,j} \right) \tag{1a}$$

$$\text{s.t. } G_{s,k+1,j} = G_{s,k,j} + \Delta G_{s,k,j},\ s \in \mathcal{N}, k \in \mathcal{T}, j \in \mathcal{G} \tag{1b}$$

$$\sum_{(i,j) \in \mathcal{L}_j} P_{s,k,i,j} + \sum_{i \in \mathcal{G}_j} G_{s,k,i} = \sum_{i \in \mathcal{D}_j} D_{s,k,i} - \sum_{i \in \mathcal{W}_j} W_{s,k,i},\ s \in \mathcal{N}, k \in \mathcal{T}, j \in \mathcal{B} \tag{1c}$$

$$P_{s,k,i,j} = b_{i,j}(\theta_{s,k,i} - \theta_{s,k,j}),\ s \in \mathcal{N}, k \in \mathcal{T}, (i,j) \in \mathcal{L} \tag{1d}$$

$$0 \le G_{s,k,j} \le x_{k,j} G_j^{max},\ s \in \mathcal{N}, k \in \mathcal{T}, j \in \mathcal{G} \tag{1e}$$

$$|\Delta G_{s,k,j}| \le \Delta G_j^{max},\ s \in \mathcal{N}, k \in \mathcal{T}, j \in \mathcal{G} \tag{1f}$$

$$|P_{s,k,i,j}| \le P_{i,j}^{max},\ s \in \mathcal{N}, k \in \mathcal{T}, (i,j) \in \mathcal{L} \tag{1g}$$

$$|\theta_{s,k,j}| \le \theta_j^{max},\ s \in \mathcal{N}, k \in \mathcal{T}, j \in \mathcal{B} \tag{1h}$$

$$x_{k,j} \in \{0,1\},\ k \in \mathcal{T}, j \in \mathcal{G} \tag{1i}$$

Here, $\mathcal{G}, \mathcal{L}$, and $\mathcal{B}$ are the sets of generators, lines, and transmission nodes (intersections of lines, known as buses) in the network in the geographical region, respectively. $\mathcal{D}_j$ and $\mathcal{W}_j$ are the sets of demand and wind-supply nodes connected to bus j, respectively. The symbol $\mathcal{N}$ denotes the set of scenarios for wind level and demand over the time horizon $\mathcal{T} := \{0, ..., T\}$. The *first stage* decision variables are the generator on/off states $x_{k,j}$ over the complete time horizon. The decision variables in each second-stage scenario s are the generator supply levels $G_{s,k,j}$ for time instant k, and bus j, the transmission line power flows $P_{s,k,j}$, and the bus angles $\theta_{s,k,j}$ (which are related to the *phase* of AC current). The random data in each scenario are the wind supply flows $W_{s,k,i}$ and the demand levels $D_{s,k,i}$ across the network. The values of $G_{s,0,j}$ and $x_{0,j}$ are fixed by initial conditions. Further modeling details can be found in previous references [8], [4].

In this work, we solve the convex relaxation of (1) obtained by replacing the binary restrictions (1i) with the constraints $0 \le x_{k,j} \le 1$. This relaxation is the first step in virtually all practical solutions of (1), it is the linear optimization problem solved at the "root node" of branch-and-bound approaches. Empirically, deterministic UC problems have been observed to have a small gap between the optimal value of the convex relaxation and the true optimal solution when combined with cutting-plane techniques and feasibility heuristics [9]. This often implies that a sufficiently small gap is obtainable with little or no enumeration of the branch-and-bound tree. Moreover, the relaxation has the same structure as stochastic ED formations; however, those need to be solved on a more stringent 5-minute time scale, which is typically coupled with scenario pruning techniques. So while we expect the performance reported here to be comparable for stochastic ED problems for given scenario set size, the actual performance needs more work for comparison due to the variability of pruning strategies.

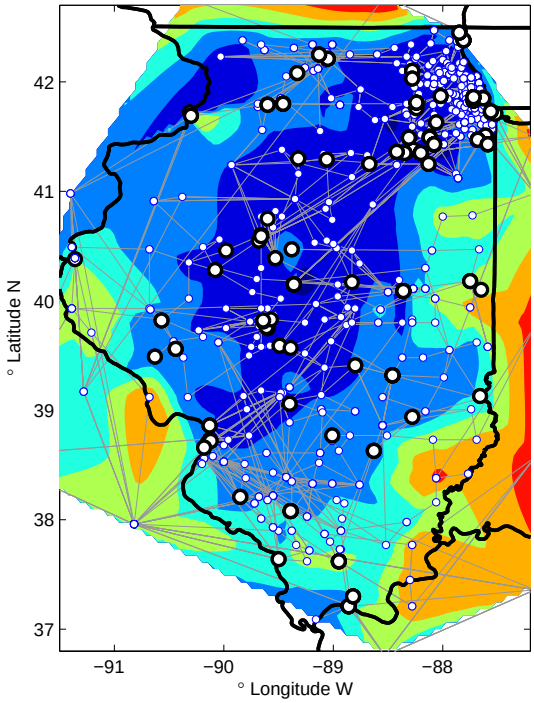

Fig. 1: Network topology and snapshot of price distribution (color map).

The convex relaxation of (1) is a very large linear optimization problem and results in a problem of the form (4). Our model incorporates the transmission network of the State of Illinois, which contains approximately 2,000 transmission nodes, 2,500 transmission lines, 900 demand nodes, and 300 generation nodes (illustrated in Figure 1). A deterministic formulation over this geographical region, when expressed over the standard $T = 24$ hours time horizon, can have up to approximately $120,000$ variables and $119,000$ constraints. These numbers are effectively multiplied by the number of scenarios, which results into problems with several billion of variables. Moreover, the size of the coupling is significant. The number of variables in the first-stage block x_0 of (2) is the number of generators times the number of time steps, leading to sizes of $6,264$, which makes parallel decomposition nontrivial.

To summarize, the following features of our problem make the development of high performance solutions such as the ones described here, highly challenging:

1) the scenario subproblem (which is the main computation per node) is sparse and unstructured, comprising of irregular network-type constraints (1b), bus-angle constraints (1c), and technological constraints (1e)-(1g) that are replicated over time.
2) the space+time+uncertainty characteristics of the problem give it effectively a "4D" flavor, and consequently, enormous instances are obtained.
3) the presence of network constraints and inequalities results in problems with saddle-point structure and changing active sets, traditional issues that hamper high performance.

III. Computational Approach

Convex quadratic two-stage stochastic optimization problems with recourse are mathematically formulated as

$$\begin{aligned} &\min_x \left(\frac{1}{2}x_0^T Q_0 x + c_0^T x_0\right) + \mathbb{E}_\xi[G(x_0,\xi)] \\ &\text{s.t. } T_0 x_0 = b_0, x_0 \geq 0, \end{aligned} \tag{2}$$

where the *recourse function* $G(x_0,\xi)$ is defined by

$$\begin{aligned} G(x_0,\xi) = \min_x \ &\frac{1}{2}x^T Q_\xi x + c_\xi^T x \\ &\text{s.t. } T_\xi x_0 + W_\xi x = b_\xi, x \geq 0. \end{aligned} \tag{3}$$

The expected value $\mathbb{E}[\cdot]$, which is assumed to be well defined, is taken with respect to the random variable ξ, which contains the data $(Q_\xi, c_\xi, T_\xi, W_\xi, b_\xi)$. The matrix Q_ξ is assumed to be positive semidefinite for all possible ξ. W_ξ, the *recourse* matrix, is assumed to have full row rank. T_ξ, the *technology* matrix, need not have full rank. The deterministic matrices Q_0 and T_0 are assumed to be positive semidefinite and of full row rank, respectively. The variable x_0 is called the *first-stage* decision, which is a decision to be made now. The *second-stage* decision x is a *recourse* or corrective decision that one makes in the future after some random event occurs. The stochastic optimization problem finds the optimal decision to be made now that has the minimal expected cost in the future.

Our computational approach focuses on solving sample average approximation (SAAs) of stochastic optimization problem (2). The SAA approach avoids the evaluation of the expectation $\mathbb{E}_\xi[G(x_0,\xi)]$, which is usually impossible or too expensive because it requires the evaluation of a highly-dimensional integral. Instead, the SAA approach replaces the expectation operator with the sample average $\sum_{i=1}^N G(x_0,\xi_i)$ computed by generating N samples $(Q_i, c_i, T_i, W_i, b_i)$ of ξ. The resulting SAA problem is a convex quadratic optimization problems with *dual block-angular* structure of the form

$$\begin{aligned} \min_{x_i, i=0,\ldots,N} \ &\frac{1}{2}x_0^T Q_0 x_0 + c_0^T x_0 + \frac{1}{N}\sum_{i=1}^N \left(\frac{1}{2}x_i^T Q_i x_i + c_i^T x_i\right) \\ \text{s.t. } \ & T_0 x_0 = b_0, \\ & T_1 x_0 + W_1 x_1 = b_1, \\ & T_2 x_0 + W_2 x_2 = b_2, \\ & \vdots \quad\quad \ddots \quad\quad \vdots \\ & T_N x_0 + W_N x_N = b_N, \\ & x_0 \geq 0, \quad x_1 \geq 0 \quad , x_2 \geq 0, \quad \ldots \quad x_N \geq 0. \end{aligned} \tag{4}$$

Our approach is to exploit the dual-block angular structure of the SAA problem (4) and parallelize the computations inside the optimization algorithm, namely Mehrotra's predictor-corrector interior-point algorithm [10]. The computational burden of the algorithm is on solving linear systems. The rest of the computations (matrix-vector multiplications and vector-vector operations) represent only a very small portion of the

computations and can be parallelized efficiently. In this section we focus the presentation on the decomposition scheme used to solve the interior-point linear systems. The reader is refered to [6], [11] for a complete description of the approach.

The interior-point linear linear systems for the SAA problem (4) have a particular form, namely the system's matrix is of the form

$$K := \begin{bmatrix} K_1 & & & B_1 \\ & \ddots & & \vdots \\ & & K_N & B_N \\ B_1^T & \dots & B_N^T & K_0 \end{bmatrix}. \tag{5}$$

Here

$$K_0 = \begin{bmatrix} Q_0 + D_0 & T_0^T \\ T_0 & 0 \end{bmatrix} \text{ and } K_i = \begin{bmatrix} Q_i + D_i & W_i^T \\ W_i & 0 \end{bmatrix}$$

are symmetric indefinite saddle-point linear systems, the 1×1 block Q_i is symmetric positive definite and the 2×2 block is zero. Matrices B_i are given by

$$B_i = \begin{bmatrix} 0 & 0 \\ T_i & 0 \end{bmatrix}.$$

Also, matrices $D_0, D_1, \dots, D_N$ are diagonal with positive diagonal entries specific to the use of interior-point methods. Some of the entries of these diagonal matrices approach zero and other remain positive and of arbitrary size as the interior-point optimization approaches the optimal solution, causing the condition number of the K_i matrices to increase unbounded. Close to optimality, linear systems can have a condition number as large as 10^{25}. This is the well-known *ill-conditioning* of interior-point methods, a complicating feature of this class of methods, which otherwise have the best known complexity and convergence properties.

Decomposition of linear systems such as K is obtained by using a Schur complement technique, which can be viewed as block Gaussian elimination of the bordering blocks of (5). First Schur complement C is computed,

$$C = K_0 - \sum_{i=1}^{N} B_i^T K_i^{-1} B_i, \tag{6}$$

and then the first-stage part of the solution is obtained by solving

$$C \Delta z_0 = r_0 - \sum_{i=1}^{N} B_i^T K_i^{-1} r_i. \tag{7}$$

Finally, the second-stage part of the solution can be obtained for all $i = 1, \dots, N$ from

$$K_i \Delta z_i = B_i \Delta z_0 - r_i. \tag{8}$$

A quick look at (6)-(8) reveals great scope for parallelism. More specifically the computation of the scenarios contributions $B_i^T K_i^{-1} B_i$ to the Schur complement, the evaluation of the residual in (7), and the solutions Δz_i, $i = 1, \dots, N$, can be performed independently. Interprocess communication occurs when assemblying the dense Schur complement matrix C based on (6) and computing the residual in (7). Solving the dense Schur complement system for z_0 from (7) is a bottleneck in this decomposition framework because the computations needs to be performed on only one node (or replicated among all nodes).

A similar decomposition approach is implemented in the state-of-the-art software package OOPS [11], which was used to solve many-stage stochastic optimization problems with 1 billion variables arising in portfolio optimization [11], [12]. The problems solved in the abovementioned work are considerably different than ours because our application has complicating coupling due to large number of first-stage variables. We also mention the work of Linderoth and Wright [13] which developed an asynchronous algorithm tailored for large heterogeneous and unreliable computational grids. We also mention PIPS-S [14] that is a parallel implementation of revised dual simplex for dual block angular problems. Alternative parallel decomposition techniques for the solution of dual angular problems are reviewed by Vladimirou and Zenios [15], however none of them have been implemented on supercomputers nor attempted to solve problems as large as ours.

Algorithm 1 lists an abstract view of our parallel implementation. The verbs "reduce" and "broadcast" describe the functionality of the MPI functions "MPI_Reduce" and "MPI_Bcast", respectively. In the subsequent sections we present our algorithmic developement that speeds up the sparse scenario computations of $B_i^T K_i^{-1} B_i$ terms and the implementation optimizations that accelerate the solves with the dense Schur complement matrix C and reduce the communication overhead.

Algorithm 1 The parallel procedure implemented in PIPS-IPM for solving the interior-point linear systems K based on the Schur complement decomposition (6)-(8)

Distribute N scenarios evenly across $\mathcal{P} = \{1, 2, \dots, P\}$ processes and let $\mathcal{N}_p$ be the set of scenarios assigned to process $p \in \mathcal{P}$.
Each process $p \in \mathcal{P}$ executes the following steps:

(factorization phase)

1.1. Factorize $L_i D_i L_i^T = K_i$ for each $i \in \mathcal{N}_p$.

1.2. Compute SC contribution $S_i = B_i^T K_i^{-1} B_i$ for each $i \in \mathcal{N}_p$.

1.3. Accumulate $C_p = -\sum_{i \in \mathcal{N}_p} S_i$. On process 1, let $C_1 = C_1 + K_0$.

2. Reduce SC matrix $C = \sum_{r \in \mathcal{P}} C_r$ to process 1.

3. Factorize SC matrix $L_c D_c L_c^T = C$ in process 1.

(solve phase)

4.1. Solve $w_i = L_i^{-T} D_i^{-1} L_i^{-1} r_i$ for each $i \in \mathcal{N}_p$.

4.2. Compute $v_p = \sum_{i \in \mathcal{N}_p} B_i^T w_i$.

4.3. On process 1, let $v_1 = v_1 + r_0$.

5. Reduce $v_0 = \sum_{i \in \mathcal{N}_p} v_i$ to process 1.

6.1. Solve $\Delta z_0 = C^{-1} v_0 = L_c^{-T} D_c^{-1} L_c^{-1} v_0$ in process 1.

6.2. Process 1 broadcasts z_0 to all other processes.

7. Solve $\Delta z_i = L_i^{-T} D_i^{-1} L_i^{-1} (B_i \Delta z_0 - r_i)$ for each $i \in \mathcal{N}_p$.

IV. Algorithmic Advances

The sparse scenario computations of $B_i^T K_i^{-1} B_i$, steps 1.1 and 1.2 in Algorithm 1, were identified in [6] to account up to 95 percent of the wall-time, being the main obstacle in solving realistically-sized stochastic power grid optimization problems in times comparable with industry's practices. Assemblying and solving with the Schur complement, steps 2, 3 and 6.1 in Algorithm 1, are the next largest two components of the execution time. In this section we address these computational tasks from both an algorithmic and implementation perspective with the specific goal of reducing their burden on the execution time.

A. Multicore sparse second-stage linear algebra

As we previously mentioned, the data is sparse and unstructured in the case of our application, therefore the computation of $B_i^T K_i^{-1} B_i$ needs to rely on *sparse* linear algebra kernels. Previously, we used off-the-shelf sparse linear solvers such as WSMP [16] and MA57 [17] to first factorize K_i as $L_i D_i L_i^T$, then perform triangular solves with the factors of K_i for each non-zero column of B_i, *i.e.*, computing $K_i^{-1} B_i$, and, finally, multiply the result from left with B_i^T.

This approach has two important drawbacks on multicore environments: i. the triangular solves with L_i and L_i^T do not scale well with the number of cores [17], being memory-bound; and ii. the sparsity of the columns of B_i is not exploited when solving $L_i X = B_i$, because off-the-shelf linear solvers treat the right-hand sides B_i as dense.

To address these limitations we revisited these computations and propose an approach that computes $B_i^T K_i^{-1} B_i$ from a partial sparse Bunch-Kaufman factorization of the augmented matrix

$$M_i = \begin{bmatrix} K_i & B_i^T \\ B_i & 0 \end{bmatrix}. \tag{9}$$

More specifically, the factorization of M_i is stopped after pivoting reaches last diagonal entry of K_i. At this point $-B_i^T K_i^{-1} B_i$ is computed and resides in the $(2,2)$ block of M_i.

Traditionally, the factorize phase has generally required the greatest portion of the total execution time. It typically involves the majority of the floating-point operations, and is computation bound. However, in this application, we have to compute $B_i^T K_i^{-1} B_i$, and memory traffic is the limiting factor. In exploiting the sparsity not only in K_i, but also in B_i we (i) significantly reduce the number of floating point operations and (ii) can use in-memory sparse matrix compression techniques to reduce the memory traffic on multicore architectures. In the numerical section we refer to this compression techniques as PARDISO-SC whereas PARDISO is the uncompressed triangular solve based on K_i. As a result, the approach PARDISO-SC is much better suitable for multicore parallelization than the triangular solves, and consequently, the speed-up over the previous approach is quite considerable, as we show in numerical experiments.

In this work, we use the augmented factorization method that is implemented in PARDISO [7]. The solver uses a static pivoting strategy [18], [19] and perturbs the matrix whenever numerically acceptable pivots cannot be found within a diagonal supernode block. This means that only a perturbation of $B_i^T K_i^{-1} B_i$ is computed and therefore an error-absorption phase is needed in order to obtain accurate solutions. We use BiCGStab Krylov space iterative method, in which the the perturbed factorization of C acts as a very efficient preconditioner [7] and keeps the number of absorption BiCGStab steps very low.

In terms of computational cost, each BiCGStab iteration requires the application of the preconditioner, which is exactly the solve phase of Algorithm 1, and the multiplication of K with the residual. The rest of the computations consist of local vector-vector operations of low cost that can be trivially parallelized.

On the implementation side this method requires a BiCGStab error-absorption loop wrapping the *solve phase* in Algorithm 1, as compactly shown in Algorithm 2. The complete details of this technique as well a specification of the BiCGStab method are given in [7].

Algorithm 2 A compact description of our BiCGStab-based technique for absorbing the pivot perturbations occurring in the sparse scenario computations. It reuses the computational components of Algorithm 1, the exact correspondence is indicated by italicized statements.

(BiCGStab initialization)
compute the preconditioner (*factorization phase of Algorithm 1*)
compute initial point (*solve phase of Algorithm 1*)
compute initial residual (*multiply with* K)

(error-absorption BiCGStab loop)
while BiCGStab has not converged **do**
  apply the preconditioner (*solve phase of Algorithm 1*)
  compute residual (*multiply with* K)
  BiCGStab iterate updates
end while

B. Intranode acceleration of dense first-stage linear algebra

We also revisit the computations needed for the first-stage solution, namely steps 3. and 6.1. in Algorithm 1. The Schur complement matrix C is dense since its computation requires the matrix inversion (see formula (6)). Consequently, storing and solving linear systems involving C is expensive. Additionally, these computations are performed on one node only; consquently, they have a considerable negative impact on both the total time-to-solution and the parallel scalability.

On BG/P we distributed matrix C and used Elemental [20] to solve the dense linear systems in parallel on all nodes. In this work we take a different route since the both XC30 and XK7 architectures have much faster nodes (both in terms of clock rate and number of cores) and more and faster memory. More specifically, we perform the first-stage computations locally and accelerate them by involving multiple cores on XC30 and GPU on XK7 in the linear solves.

On XC30, we use the LAPACK routines *dsytrf* (for the symmetric indefinite factorization of C) and *dsytrs* (for the

triangular solves with the factors of C). We use Cray Scientific Library (cray-libsci) implementation for both BLAS and LAPACK. The BLAS implementation is multithreaded and compatible with OpenMP.

The presence of the NVIDIA Tesla K20 GPU accelerator on XK7 nodes is an opportunity to speed up the first-stage computations, since GPUs are very suitable for dense linear algebra operations. In our implementation we use routines from MAGMA [21]. It offers an interface similar to LAPACK for GPUs. We used the factorization routine *magma_dgetrf_gpu* and triangular solve routine *magma_dgetrs_gpu* for general square matrices since MAGMA 0.2, which is the latest version at this moment, does not have a GPU implementation for the factorization of symmetric indefinite matrices such as C. Such implementation is capable to halve the solution time by exploiting the symmetry.

C. Optimized communication pattern

The communication strategy presented in Algorithm 1 is slightly different from our previous BG/P implementation reported in [6]. On BG/P we performed "AllReduce" in steps 2 and 5 of Algorithm 1 since on BG/P MPI_Allreduce is much faster than MPI_reduce (this anomaly is likely to be caused by an implementation problem of BG/P's MPI_reduce). Consequently, the dense factorization of C and the solves with it (steps 3 and 6.1 in Algorithm 1) were replicated on all nodes on BG/P. In this work we use MPI_reduce and assemble the Schur complement to MPI rank 0 only. The dense computations are carried out in this process alone. The solution is then broadcasted (MPI_Bcast) to all other MPI ranks. This is step 6.2 in Algorithm 1. This change in the communication-computation pattern is about twice faster on both XC30 and XK7 systems than the previous implementation. It is also more efficient with respect to energy consumption since it does not replicate computations unnecessarily.

V. Numerical experiments

In this section we describe the experiments and investigate the performance of our computational advances by looking at several performance metrics and indicators such as intranode thread affinity, weak and strong scaling, and sustained peak performance. We also identify the bottlenecks of the decomposition scheme discuss their impact on the performance. Finally, we report on the time-to-solution for a 24-hour horizon unit commitment problem with $16,384$ scenarios, 1.95 billion variables and 1.947 billion constraints.

A. The architectures and the software environments

We use two architectures, Cray XK7 and Cray XC30. The Cray XK7 compute nodes offers a hybrid environment, having the AMD's "Interlagos" Opteron 6200 Series 64-bit processor and the NVIDIA's Tesla K20x GPU Accelerator. The CPU has 16 cores distributed on 8 Bulldozer modules and 32GB DDR3 main memory. The K20x chip has one Kepler GK110 GPU with $2,688$ cores and 6GB GDDR5 main memory. The connection between nodes is done using a Gemini interconnect through HyperTransport 3.0 technology. We use two deployments of this architecture, "Todi" from Swiss National Supercomputing Centre, which has 272 nodes and "Titan" machine from Oak Ridge National Laboratory, which has $18,688$.

TABLE I: Average times per scenario needed for the sparse scenario computation of $B_i^T K_i^{-1} B_i$ on (a) a XK7 node and (b) on a XC30 node with various number of threads and MPI processes per node. The scenarios come from a 24-hour horizon problem.

# of MPI ranks	(a) Number of threads per process on XK7					
	1	1	2	4	8	16
	PARDISO	PARDISO-SC				
1	356.5	41.1	25.6	17.5	14.4	15.8
2	459.6	41.1	26.5	20.5	21.1	
4	506.6	43.3	32.6	34.4		
8	712.5	55.1	61.0			
16	707.9	113.7				

# of MPI ranks	(b) Number of threads per process on XC30					
	1	1	2	4	8	16
	PARDISO	PARDISO-SC				
1	168.3	19.0	12.9	8.4	6.2	5.9
2	175.3	18.9	12.4	8.5	7.4	
4	194.6	19.5	13.8	12.1		
8	284.2	22.1	21.8			
16	281.6	38.5				

The Cray XC30 nodes are based on the Intel's "Sandy-Bridge" Xeon E5-2600 2×8-core 64-bit processor, each node having 32GB of memory. The interconnect uses dragonfly topology and Aries network chips. We have used the $2,254$-nodes machine "Piz Daint" of Swiss National Supercomputing Centre.

In all experiments we have used GNU compilers and Cray Scientific Library implementation of BLAS and LAPACK. On XK7 machines we have used MAGMA 0.2 and CUDA Toolkit 5.0. The FLOPS evaluation was done using Cray Performance Analysis Tool (CrayPAT).

B. Intranode scaling and thread affinity

We investigate the behaviour of the new linear algebra developments presented in Section IV-A and IV-B on multicore nodes.

We first tested different assignments of processes and threads to physical cores and identified the core layouts that makes the best use of memory hierarchy during our sparse and dense computations. For XK7, we found that it is always optimal to assign cores residing in the same NUMA region to the same MPI process. On XC30 it is similar, in the sense that it is optimal to keep MPI processes inside one of the two 8-core processors. Multithreading is quite different on the

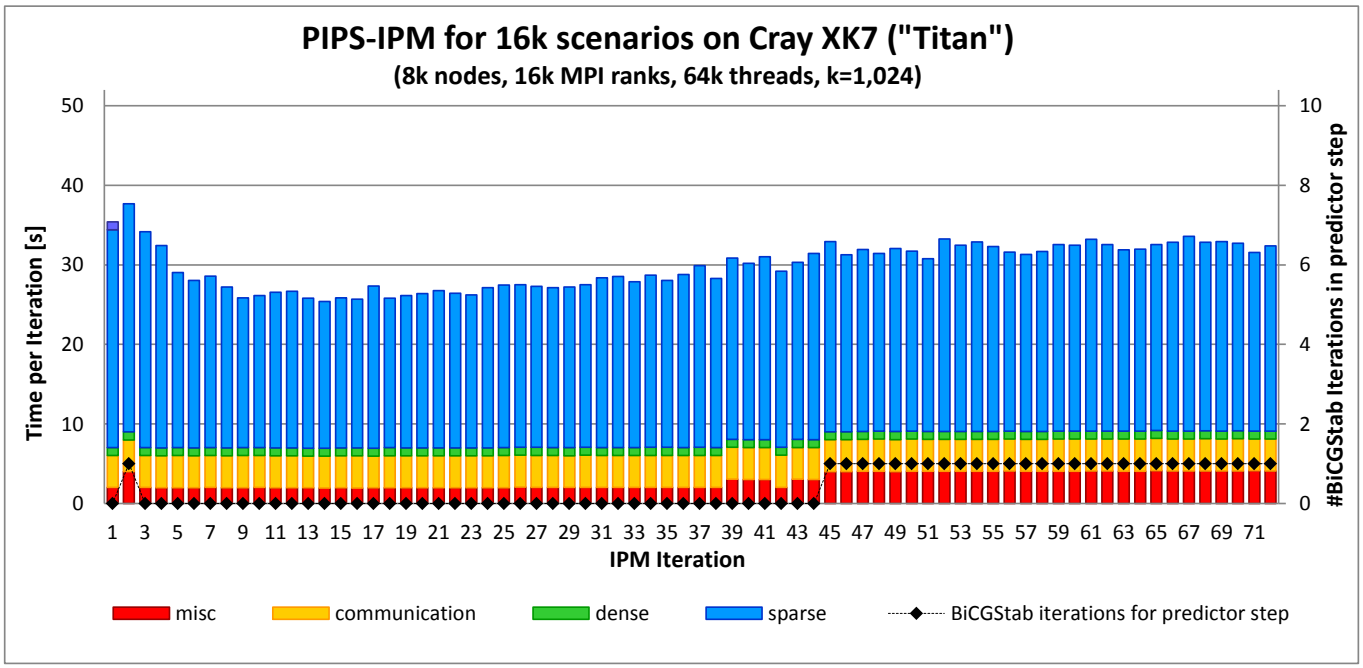

Fig. 2: Breakdown of the execution time for each iteration of PIPS-IPM when solving a 24-hour horizon problem with $16,384$ scenarios on $8,192$ nodes of "Titan" ($16,384$ MPI ranks).

two systems. On XK7 there is no speed-up from running two threads on the same bulldozer module. Our diagnosis tests indicate that the cray-libsci BLAS, used both by LAPACK and PARDISO-SC, does not share the Bulldozer module's 256-bit AVX floating point unit and the two threads runs sequentially. This is a limiting factor in the performance of our code on XK7 and we currently try to resolve this issue. On XC30, multithreading is normal even in the extreme case when the processors are saturated with threads.

Then we experiment with the number of MPI processes per node and number of threads per MPI process and determine the combination that gives the best execution times per scenario. Table I displays the average execution times needed by PARDISO-SC to compute $B_i^T K_i^{-1} B_i$ on XK7 and XC30. For reference, we also show the times of the "backsolve" approach that we used in PIPS-IPM prior to this work to compute $B_i^T K_i^{-1} B_i$, for which we have used PARDISO as an off-the-shelf linear solver. We note that the augmented incomplete factorization ("PARDISO-SC") times are significantly smaller than the "backsolve" ("PARDISO") times.

It can be also seen in Table I that PARDISO-SC scales reasonably well with the number of threads on both systems, with the exception of XK7, there is no benefit of saturating the node with threads, as we pointed out in the previous paragraph.

In term of FLOPS per node, PARDISO-SC is capable of achieving about 25% of the node's peak when used with 16 threads. These numbers includes all solution phases including nested dissection based on METIS 5.1, structural and numerical factorization and triangular solves. For the layout we use in the large runs, that is 4 threads and 4 MPI processes, PARDISO-SC obtained 1.61 GFLOPS per XC30 core and 15.5% of the XC30 node's peak. This layout means that four of the $B_i^T K_i^{-1} B_i$ terms are computed on a node in the same time. On XK7, PARDISO-SC attains 1.17 GFLOPS per core when running 4 MPI processes, each with 2 threads, which is about 14.6% of the core peak.

The dense factorization performed on XK7 using GPU and Magma took approximately 0.7 seconds, which is indicates to about 234 GFLOPS considering the complexity of a LU factorization (we have not used CrayPAT since it does not instrument CUDA code yet). As another comparison, the same matrix can be factorized on XK7's CPU using LAPACK with 4 threads as fast as 5.6 seconds. On XC30, the fastest LAPACK dense factorization can be performed in about 4 seconds using 4 threads and achieves 3.89 GFLOPS per core, 37.4% of a core's peak.

C. Large scale simulations

In this section we report on the large scale performance of PIPS on "Titan" XK7 and "Piz Daint" XC30. We report strong scaling efficiency, which means that a 24-hour horizon problem with $16,384$ scenarios was solved with increasingly large number of nodes. This instance has a total of 1.95 billion variables and 1.947 billion constraints. Figure 3 show the wall-time of one interior-point iteration, wall-times of the computational sub-components and their ideal speed-ups obtained on the two systems. On "Titan" we used 2 MPI processes per node, each running 4 threads on 4 bulldozer modules in the same NUMA region. On "Piz Daint" we used 4 MPI processes per node, each each running 4 threads. The

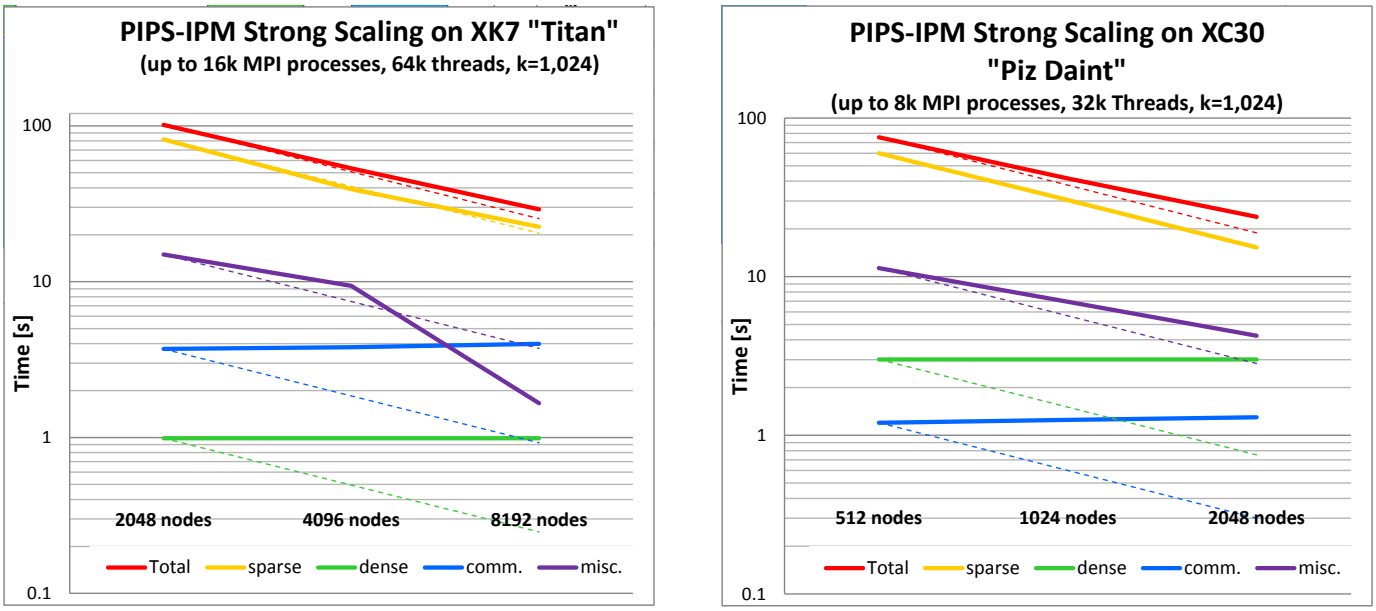

Fig. 3: Strong scaling plots for "Titan" (left) and "Piz Daint" (right) for a 24-hour horizon problem with $16,384$ scenarios. "Total" shows the time for one iteration averaged over the first 15 iterations. We also show execution times for the computational sub-components: sparse scenario computations of $B_i^T K_i^{-1} B_i$ by "sparse", dense factorization by "dense" and communication by "comm.". The label "misc" refers both to time spent in the error-absorption phase as well as load imbalance. Dotted lines show ideal speed-ups.

parallel efficiency of PIPS-IPM on "Titan" is 87.0% (from $2,048$ to $8,192$ nodes) and on 'Piz Daint" is 79.2% (from 512 to 2048 nodes). The better parallel efficiency on "Titan" despite a much large number of nodes is partially due to the fact that the main computational bottleneck, *i.e.*, the dense first-stage factorization, is performed very fast on GPU.

The efficiencies on both machines are very high and close to what we have previously observed on "Intrepid" BG/P in [6], despite the acceleration of the intranode sparse scenario computations and the slower interconnects that both "Titan" and "Piz Daint" have. The one order of magnitued speed-up of the sparse computations obtained given by the incomplete augmented factorization algorithm would have significantly reduced the parallel efficiency without our implementation developments that accelerate the dense computations and reduce communication overhead.

On "Titan" we performed a weak scaling study, in which we varied both the number of scenarios and nodes, keeping a ratio of 2 scenarios per node (one scenario per MPI process). The largest problem has $32,768$ scenarios (3.90 billion variables and 3.895 billion constraints) and has been solved on $16,384$ nodes (87.6% of the machine). For this problem the size of IPM linear system K is 7.8 billion. Figure 4 shows time per iteration (average of the first 15 IPM iterations) and a breakdown of the execution time for runs on $2,048$, $4,096$, $8,192$, and $16,384$ nodes. The parallel efficiency that is also shown in Figure 4 is computed using the $2,048$-node run as reference and, for the largest run being 97.7% percent. We note that these are times per iteration and therefore they show the efficiency of linear algebra developments, not of the entire optimization solver. For similarly sized problems solved on BG/P [7] we observed that the total number of IPM iterations increases with about 10% each time the number of scenarios is doubled. Based on this, we extrapolate that the efficiency of the $16,384$ nodes simulation in terms of time-to-solution would be in the vecinity of 72%.

D. Time-to-solution and peak performance considerations

We report wall-time needed to solve the 24-hour horizon problem with $16,384$ scenarios to optimality. Our stopping criteria consist of duality gap less than 10^{-6} and residual norm less than 10^{-10}. The former criterion indicates that the objective value found by PIPS-IPM is within 6 significant digits close to the optimal value, while the latter shows that the constraints of the problem are satisfied within a 10^{-10} tolerance.

A total of 72 interior-point iterations was needed to solve the problem. This was done in 37.54 minutes on "Titan" and in 29.73 minutes on "Piz Daint". The runs on the two systems are the largest runs used in the strong scaling plots presented in Figure 3. We also show in Figure 2 the breakdown of the execution time for each interior-point iteration and the number of BiCGStab error-absorption steps taken when computing the predictor directions (similar number of BiCGStab steps is needed for corrector directions).

We note in particular in Figure 2 that the well-known ill-conditioning of interior-point linear systems that occurs near the optimal solution does not increase the number of BiCGStab steps significantly. This indicates that the pivot

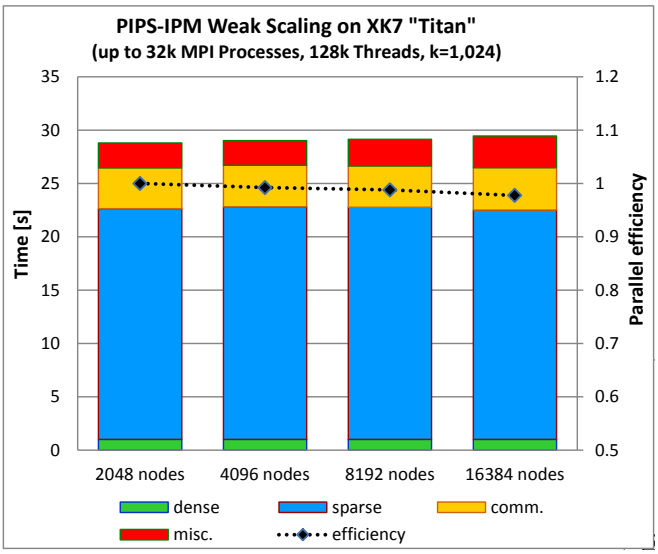

Fig. 4: Weak scaling plot for "Titan" when solving 24-hour horizon instances with 2 scenarios per node. Times shown are for one iteration averaged over the first 15 IPM iterations. We also a breakdown of the execution time. Besides the error-absorption time, "misc." also contains load imbalance. Dotted line shows observed parallel efficiency, with the smallest run acting as reference.

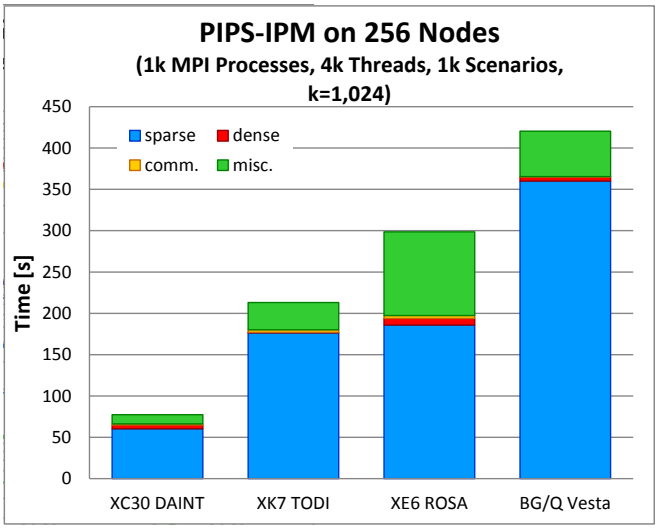

Fig. 5: Execution times per PIPS-IPM iteration on some of the most popular high-performance architectures. A 24-hour horizon problem with $1,024$ scenarios was solved on 256 nodes of each system, using 4 threads per MPI process.

perturbations remain small when ill-conditioned linear systems are encountered, a sign that the pivoting techniques inside PARDISO-SC are numerically stable.

The "real-time" execution time we obtained on the two systems are a consequence of a significant increase in FLOPS rate. On "Piz Daint" XC30 CrayPAT reported 12.5% of the peak when solving a problem with $8,192$ scenarios on 256 nodes (4 MPI ranks per node, 4 threads per process). FLOPS peak evaluation for a larger number of nodes was impossible, CrayPAT caused PIPS-IPM to hang. However, from the strong scaling plot, one can infer that the peak performance on full machine is about 10%. On XK7 such evaluation is not possible with CrayPAT when CUDA code is enabled in PIPS-IPM. One can extrapolate, based on the intranode performance of PIPS-SC reported in Section V-B and the strong scaling efficiency reported in Section V-C, that PIPS-IPM achieves on XK7 a sustained CPU peak rate of the same order as on XC30, if the base theoretical CPU peak for XK7 is calculated with one core per bulldozer module.

The peak performance of 10% is at least acceptable in our opinion, given the irregular memory access patterns of the sparse linear algebra and the amount of integer operations required by sparse factorizations and not included in FLOPS evaluation. Direct factorizations of sparse matrices of similar sizes as ours on supercomputers are reported in [16], where it is reported a sustained peak performance similar to our, approximately 12.6% of the peak (7.05 Teraflops on $4,096$ BG/P nodes). The authors of this work alo note this is the highest known FLOPS rate for this class of matrices and these class of computing platforms.

In addition, we mention that our novel algorithmic developments of the sparse linear algebra computations within PARDISO-SC enabled us to obtain a factor of over 10 increase in the FLOPS rate on every single core over the previous version of PIPS-IPM. Before this work, PIPS-IPM was capable of achieving a little bit more that 1% of the peak on a BG/P platform [6].

The modification of the computation-communication strategy we have done in this work, which causes only one node to be involved in the first-stage computations, is a limiting factor in achieving better peak performance. The alternative communication pattern, which we used on BG/P, would increase the FLOPS rate but will perform poorer with respect to at least two metrics: time-to-solution (due to communication overhead) and energy consumption (due to replicating computations over all nodes). In our opinion, this shows that peak performance should be seen as a less relevant performance metric in the case of our decomposition pattern and application.

Finally we present in Figure 5 a comparison of the computational cost of a IPM iteration and of the computational subcomponents of PIPS-IPM on two additional high-performance computers, Cray XE6 "Rosa" ($1,496$ nodes) from Swiss National Supercomputing Centre and IBM BG/Q "Vesta" (1 cabinet, $2,048$ nodes) of Argonne National Laboratory. XE6 systems are similar to XK7, having two AMD "Interlagos" processors per node (but no GPUs) and use a Gemini 3D interconnect. BG/Q has 16 PowerPC A2 cores operating at 1600Mhz per node and uses a 5D torus interconnect. The comparison of these systems is meant to provide a guideline on the role played by hardware on the solution times. In particular we note considerably longer execution times on BG/Q due to slower cores, an important reduction in the dense computations on XK7 over the similar XE6 due to the use of

GPUs and a very fast interconnect on BG/Q. XC30 is the fastest platform by a large margin due to the performance of "SandyBridge" XEONs. In the case of our application, where virtually hundreds of gigawatts are dispatched every day based on UC and ED models, the high energy per flops ratio of XC30 platforms may be less relevant and systems of this type look appealing from an operational perspective.

VI. Conclusions

In this work we adressed the issue of solving complex energy systems optimization problems with thousands of uncertain scenarios in times comparable to the ones required by the operational practices of power grid operators. For this we have proposed and implemented in PARDISO a novel multi-threaded restructuration of the sparse linear algebra second-stage computations of PIPS-IPM. This approach makes very efficient use of the multi-core nodes, gives a drastic reduction in the execution times and makes possible "real-time" solution to our power grid optimization problems. We also addressed and considerably reduced the bottlenecks of our decomposition scheme, namely the first-stage dense factorizations and the communication, which translated in very good parallel efficiencies on thousands of nodes of two modern supercomputers.

Several other improvements will be required before possible deployment, and that includes refining the solution to obtain a good integer one for UC problems and exploiting within-scenario parallelism for ED problems. These are both non-trivial tasks, however, we point out that solving the relaxation is an essential first step to get to that level of performance for UC, and several heuristics exist to refine the solution to a good integer one. Moreover, the massive economic importance of solving this problem in a timely fashion coupled with the stochasticity prompted by renewable energy imperatives makes this first step, where we achieve timings compatible with the real time requirements of the operators, a very important one.

Acknowledgments

This research used resources of the Laboratory Computing Resource Center and the Argonne Leadership Computing Facility (ALCF) at Argonne National Laboratory, which is supported by the Office of Science of the U.S. Department of Energy under contract DE-AC02-06CH11357. Computing time on the "Titan" was provided by the Oak Ridge Leadership Computing Facility at Oak Ridge National Laboratory, which is supported by the Office of Science of the U.S. Department of Energy under Contract DE-AC05-00OR22725. Computing time on the Swiss National Supercomputing Center's "Rosa", "Todi" and "Piz Daint" was provided by a USI-CSCS allocation contract. Computing time on "Intrepid" was granted under the 2012 DOE INCITE Award "Optimization of Complex Energy Systems under Uncertainty," PI Mihai Anitescu, Co-PI Cosmin G. Petra. Computing time on "Vesta" was available through ALCF Director's Discretionary Allocation program.

References

[1] Federal Energy Regulatory Commission, "Electric power markets: National overview," Webpage, http://www.ferc.gov/market-oversight/mkt-electric/overview.asp, retrieved April 2013.

[2] M. Shahidehpour, H. Yamin, and Z. Li, *Market Operations in Electric Power Systems: Forecasting, Scheduling, and Risk Management*. Wiley, New York, 2002.

[3] M. Carrion, A. Philpott, A. Conejo, and J. Arroyo, "A stochastic programming approach to electric energy procurement for large consumers," *IEEE Transactions on Power Systems*, vol. 22, no. 2, pp. 744–754, may 2007.

[4] E. Constantinescu, V. Zavala, M. Rocklin, S. Lee, and M. Anitescu, "A computational framework for uncertainty quantification and stochastic optimization in unit commitment with wind power generation," *IEEE Transactions on Power Systems*, vol. 26, no. 1, pp. 431–441, feb. 2011.

[5] G. Pritchard, G. Zakeri, and A. Philpott, "A single-settlement, energy-only electric power market for unpredictable and intermittent participants," *Oper. Res.*, vol. 58, no. 4–2, pp. 1210–1219, July 2010.

[6] M. Lubin, C. G. Petra, M. Anitescu, and V. Zavala, "Scalable stochastic optimization of complex energy systems," in *Proceedings of 2011 International Conference for High Performance Computing, Networking, Storage and Analysis*, ser. SC '11. New York, USA: ACM, 2011, pp. 64:1–64:64.

[7] C. G. Petra, O. Schenk, M. Lubin, and K. Gäertner, "An augmented incomplete factorization approach for computing the Schur complement in stochastic optimization," Preprint ANL/MCS-P4030-0113, Argonne National Laboratory, Tech. Rep., 2013.

[8] M. Lubin, C. G. Petra, and M. Anitescu, "The parallel solution of dense saddle-point linear systems arising in stochastic programming," *Optimization Methods and Software*, vol. 27, no. 4–5, pp. 845–864, 2012.

[9] D. Streiffert, R. Philbrick, and A. Ott, "A mixed integer programming solution for market clearing and reliability analysis," in *Power Engineering Society General Meeting, 2005. IEEE*, June 2005, pp. 2724–2731 Vol. 3.

[10] S. Mehrotra, "On the implementation of a primal-dual interior point method," *SIAM Journal on Optimization*, vol. 2, no. 4, pp. 575–601, 1992.

[11] J. Gondzio and A. Grothey, "Exploiting structure in parallel implementation of interior point methods for optimization," *Computational Management Science*, vol. 6, no. 2, pp. 135–160, May 2009.

[12] ——, "Direct solution of linear systems of size 10^9 arising in optimization with interior point methods," in *PPAM*, 2005, pp. 513–525.

[13] J. Linderoth and S. Wright, "Decomposition algorithms for stochastic programming on a computational grid," *Comput. Optim. Appl.*, vol. 24, no. 2-3, pp. 207–250, 2003.

[14] M. Lubin, J. Hall, C. Petra, and M. Anitescu, "Parallel distributed-memory simplex for large-scale stochastic lp problems," *Computational Optimization and Applications*, pp. 1–26, 2013. [Online]. Available: http://dx.doi.org/10.1007/s10589-013-9542-y

[15] H. Vladimirou and S. Zenios, "Scalable parallel computations for large-scale stochastic programming," *Annals of Operations Research*, vol. 90, pp. 87–129, 1999.

[16] A. Gupta, S. Koric, and T. George, "Sparse matrix factorization on massively parallel computers," in *Proceedings of the Conference on High Performance Computing Networking, Storage and Analysis*, ser. SC '09. New York, NY, USA: ACM, 2009, pp. 1:1–1:12.

[17] J. Hogg and J. Scott, "A note on the solve phase of a multicore solver," SFTC Rutherford Appleton Laboratory, Tech. Rep. RAL-TR-2010-007, June 2010.

[18] O. Schenk and K. Gartner, "On fast factorization pivoting methods for symmetric indefinite systems," *Elec. Trans. Numer. Anal.*, vol. 23, pp. 158–179, 2006.

[19] O. Schenk, A. Wächter, and M. Hagemann, "Matching-based preprocessing algorithms to the solution of saddle-point problems in large-scale nonconvex interior-point optimization," *Computational Optimization and Applications*, vol. 36, no. 2, pp. 321–341, 2007.

[20] J. Poulson, B. Marker, and R. A. van de Geijn, "Elemental: A new framework for distributed memory dense matrix computations (FLAME Working Note #44)," Institute for Computational Engineering and Sciences, The University of Texas at Austin, Tech. Rep., Jun. 2010.

[21] F. Song, S. Tomov, and J. Dongarra, “Enabling and scaling matrix computations on heterogeneous multi-core and multi-gpu systems,” in *6th ACM International Conference on Supercomputing*, ser. (ICS 2012). New York, NY, USA: ACM, 2012.